Nested-Loop RAJA Extensions for Deterministic Transport

DOE Centers of Excellence Performance Portability Meeting

April 19, 2016

Adam J. Kunen
Kripke is a research tool that is informing the development of ASC codes

Internal Research and Development

Kripke v1.0
L, L+, Sweep (Aug 2014)

Kripke v1.1
L, L+, Sweep, Source, Scattering (Sept 2015)

ARDRA

Porting to RAJA during FY16/FY17

Nested-Loop RAJA
Developing RAJA extensions for use in ARDRA (Ongoing)

External PM Research/Development Collaborations

OCCA
Demoed OKL DSL (David Medina 2015)

RAJA (L2)
Demoed OMP/CUDA Portability (FY15)

CORAL CoE
Exploring CUDA algorithms Ongoing

Legion
Ongoing (Sam White, UIUC)

STAPL
Ongoing (TAMU)

Charm++
Ongoing (Sam White, UIUC)

Research with Kripke motivated nested-loop abstractions
Kripke demonstrates performance is sensitive to data layout, architecture and problem size

Data layout and loop interchange affect performance

... so do problem dimensions and architecture

Abstracting data layout and loop order enable performance portability
Discrete ordinates transport needs a portable PM that treats multidimensional loops and data as first class citizens

- Sn Transport is dominated by nested loops
  - High dimensional phase space
  - Often loops are nested 2 to 5 deep (sometimes even more)
  - Many of our loops are **perfectly nested**
  - Complex iteration patterns (sweeps, multi-material)

- Kripke shows performance is sensitive to:
  - Data layout, Loop-nesting order
  - Architecture, Compiler (vendor and version)
  - Problem specifications (zones, groups, directions, moments, etc.)

- Given Architecture+Compiler+Problem:
  - Choose Data Layout and Loop-Nest Order
  - Choose execution policies

- Preparation for Sierra requires porting to GPU
  - CUDA? OpenMP?
  - What about exascale? What PM will we need then?
Why do we need RAJA::forallN instead of just nesting the existing RAJA::forall?

Nested Loop Constructs
- Nesting RAJA::forall’s work, but don’t enable complex loop transformations
- Hard-wires code for specific patterns
- CUDA kernels require building a thread+block index space from multiple loop indices... very difficult with nested forall’s
- Nested lambdas are problematic for OMP4 and CUDA

RAJA::forall<policy_i >( IndexSet_I, [=] (int i){
    RAJA::forall<policy_j >( IndexSet_J, [=] (int j){
        RAJA::forall<policy_k >( IndexSet_K, [=] (int k){
            y[ i*nj + j ] += a[k] * x[ j*nk + k ] ;
        })
    })
});

Multi-Dimensional Arrays
- Manual array calculations are error prone
- Hard-wires code for specific data layouts

Nested loops and multidimensional arrays are needed in addition to RAJA
Nested-Loop RAJA extensions provide multidimensional support

- Maintains the RAJA philosophy
  - Separate concepts of loop execution, iteration patterns and loop bodies
  - Minor code structure changes
  - Allow incremental transition to RAJA
  - Leverage existing RAJA code (forall, IndexSet, etc.)
    - Basic nested-loops are functionally equivalent to nested RAJA::forall()’s

- Arbitrary Dimensionality
  - RAJA::forallN for any N
  - Using variadic template meta-programming, no code gen *(almost)*

- Loop Transformations (for perfectly nested loops)
  - Loop interchange
  - Multi-level tiling/blocking
  - Mapping to CUDA threads and blocks
  - Loop collapsing (OpenMP collapse(N))

- Data Layouts
  - Arbitrary data striding orders

- Portability (and hopefully performance)
  - Sequential, SIMD, OpenMP (CPU/GPU), CUDA

RAJA is a good starting point for an Sn programming model
How do we extend the RAJA::forall abstraction for single loops to nested loops?

Kripke v1.1: LTimes for DGZ layout

double *ell_ptr;
double *psi_ptr;
double *phi_ptr;

for(int nm = 0; nm < num_moments; ++nm){
    double * ell_nm = ell + nm*num_loc_dir;
    double * __restrict__ phi_nm =
        phi + nm*num_gz;

    for (int d = 0; d < num_loc_dir; d++) {
        double * __restrict__ psi_d =
            psi + d*num_loc gz;
        double ell_nm_d = ell_nm[d];

        for(int gz = 0; gz < num_loc.gz; ++ gz){
            phi_nm[gz] += ell_nm_d * psi_d[gz];
        }
    }
}

Issues with this coding style:

- Loop-nest order is fixed
  — Loop-interchange requires rewrite
- Inner 2-loops collapsed
  — An arch-specific optimization
- Fixed layout of each variable
- No obvious mapping to CUDA
  — Need to rewrite
- Code is just downright ugly

Hand coded loops are inflexible and non-portable
Nested-Loop RAJA abstracts nested loops, loop interchange, and data layouts

Kripke v1.1: LTimes for DGZ layout

define *ell_ptr;
define *psi_ptr;
define *phi_ptr;

for(int nm = 0; nm < num_moments; ++nm){
    define * ell_nm = ell + nm*num_loc_dir;
    define * __restrict__ phi_nm =
        phi + nm*num_gz;

    for (int d = 0; d < num_loc_dir; d++) {
        define * __restrict__ psi_d =
            psi + d*num_locgz;
        define ell_nm_d = ell_nm[d];

        for(int gz = 0; gz < num_loc_gz; ++gz){
            phi_nm[gz] += ell_nm_d * psi_d[gz];
        }
    }
}

Kripke+RAJA: LTimes for all layouts

define View_Psi psi(psi_ptr, ...);
define View_Phi phi(phi_ptr, ...);
define View_Ell ell(ell_ptr, ...);

forallN<NestedPolicy, NM, Dir, Group, Zone>
(is_moment, is_dir, is_group, is_zone,
[=](NM nm, Dir d, Group g, Zone z){
    phi(nm, g, z) += ell(d, nm) * psi(d, g, z);
});

Nested-loop abstraction promotes flexibility, while maintaining code structure
forallN extends RAJA concepts needed for nested-loops

forallN() abstracts an N-nested loop

- Exec policies for each loop nest
- Loop transformations

IndexSet for each loop nest

RAJA::forallN< NestedPolicy, NM, Dir, Group, Zone>(
    is_moment, is_dir, is_group, is_zone,

    [](NM nm, Dir d, Group g, Zone z){

        phi(nm, g, z) += ell(d, nm) * psi(d, g, z);
    }
);

Loop-body

Views abstract data access and provide type safety

Type-safe indices

Extensions provide nested loop concepts and promote code correctness
Execution policies enable rapid testing of diverse parallelization strategies

Sequential Policy:

```cpp
using Pol = NestedPolicy<ExecList<seq_exec, seq_exec, seq_exec, seq_exec>>;
```

Parallel region, with OpenMP threading over groups:

```cpp
using Pol = NestedPolicy<ExecList<seq_exec, seq_exec, omp_for_nowait_exec, seq_exec>,
                         OMP_Parallel<
                            Permute<PERM_KIJJ>
                         >;
```

Parallel region, with OpenMP threading over groups, collapse(2), tiling zones by 512:

```cpp
using Pol = NestedPolicy<ExecList<seq_exec, seq_exec, ompCollapseNowait_exec, ompCollapseNowait_exec>,
                         OMP_Parallel<
                            Tile< TileList<tile_none, tile_none, tile_none, tile_fixed<512>>,
                            Permute<LKJI>
                         >;
```

CUDA policy, mapping groups to threads and block in X (with 32 threads/block), and zones to blocks in Y.

```cpp
using Pol = NestedPolicy<
    ExecList<seq_exec,
              seq_exec,
              cuda_threadblock_x_exec<32>,
              cuda_block_y_exec>
>;
```

Complex loop nesting constructs are easy to implement without kernel changes
Kernel performance depends on choosing the execution policy that matches the architecture

DGZ  Dirs/Sets: 96/8  Grp/Sets: 32/1  Zones: 8x8x8, 10x10x10, ...

(Git Hash: d3799680b5fe930423a29e0478329d2edaa2e8a7)

Grind Time for DGZ LTimes Kernel in Kripke for Various Execution Policies

Performance can be tuned with policies, and w/o modifying kernels
Conclusion

- Nested Loop constructs are now officially in RAJA
  - Offload to threads (OpenMP) and GPU (CUDA)
  - Complex loop transformations are possible without impacting code

- CoE interactions have been crucial
  - Vendor cooperation has been good
  - Intel machines are looking good...
    - Optimization improvements would make marked improvement
    - OpenMP kernels in Vtune are problematic
  - IBM and NVidia (nvcc) have most issues to work out
    - Starting to explore solutions, may impact implementation details

- Starting to explore implementation in ARDRA
  - Starting with concepts that seem *less risky* and incorporating them into ARDRA
  - Just starting to move to C++11 (Sequoia+XLC only blocker)
  - Hoping that we get more vendor issues worked out soon

- Questions?